Review Article

A Survey of $k$ Nearest Neighbor Algorithms for Solving the Class Imbalanced Problem

Bo Sun$^{1}$ and Haiyan Chen$^{2}$

$^{1}$Department of Computer Science and Technology, Shandong Agricultural University, China
$^{2}$College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China

Correspondence should be addressed to Bo Sun; sunbo87@sdau.edu.cn and Haiyan Chen; chenhaiyan@nuaa.edu.cn

Received 17 January 2021; Accepted 18 February 2021; Published 3 March 2021

Academic Editor: Junwu Zhu

$k$ nearest neighbor ($k$NN) is a simple and widely used classifier; it can achieve comparable performance with more complex classifiers including decision tree and artificial neural network. Therefore, $k$NN has been listed as one of the top 10 algorithms in machine learning and data mining. On the other hand, in many classification problems, such as medical diagnosis and intrusion detection, the collected training sets are usually class imbalanced. In class imbalanced data, although positive examples are heavily outnumbered by negative ones, positive examples usually carry more meaningful information and are more important than negative examples. Similar to other classical classifiers, $k$NN is also proposed under the assumption that the training set has approximately balanced class distribution, leading to its unsatisfactory performance on imbalanced data. In addition, under a class imbalanced scenario, the global resampling strategies that are suitable to decision tree and artificial neural network often do not work well for $k$NN, which is a local information-oriented classifier. To solve this problem, researchers have conducted many works for $k$NN over the past decade. This paper presents a comprehensive survey of these works according to their different perspectives and analyzes and compares their characteristics. At last, several future directions are pointed out.

1. Introduction

$k$ nearest neighbor ($k$NN) [1] has simple implementation and supreme performance and can achieve comparable performance with more sophisticated classifiers including decision tree [2], artificial neural network [3], and support vector machine [4]. Therefore, $k$NN has been listed as one of the top 10 algorithms in machine learning and data mining [5, 6]. $k$NN has been utilized in many applications, such as pattern recognition [7], feature selection [8], and outlier detection [9]. For a test example with unknown class label, $k$NN makes a decision by employing the local information surrounding the test example. Concretely, $k$NN first simply stores all the training examples; then, in the classification phase, it takes the class occurring most frequently in the $k$ ($k \geq 1$) nearest training examples of the test example as the classification result. That is, $k$NN makes a decision according to the class distribution characteristics in the $k$ neighborhood of a test example.

Nowadays, machine learning and data mining techniques are widely used in many aspects of the information society. However, for some applications such as medical diagnosis [10], system intrusion detection [11], and network fraud detection [12], the collected training example set is usually class imbalanced, i.e., there is a large difference among the sizes of different classes. For instance, in medical diagnosis data, the majority examples are descriptions of normal patients (negative examples), and only a small proportion of examples are representatives of special patients suffering a rare disease (positive examples). But if a special patient is erroneously classified as a normal patient, the best treatment time will be missed and serious consequences will be caused.
For computer network intrusion detection data, the majority examples denote the normal access data (negative examples) and only the minority examples denote the illegal access data (positive examples). Similarly, misclassifying illegal access as a legal one will lead to the disclosure of a unit’s inner data or the steal of bank account information. From the above two instances, it can be seen that, in class imbalanced data, although the positive class is heavily outnumbered by the negative class, the positive class is usually the one in which we are more interested and is more important than the negative class. The positive class is also named the minority class while the negative class is also called the majority class.

Similar to classical classifiers such as decision tree, artificial neural network, and support vector machine, \(k\)NN is also proposed based on the assumption that a training set has approximately balanced class distribution, i.e., the classes have roughly the same number of training examples. In addition, these algorithms all employ the overall classification accuracy as the optimization objective in the classifier training phase, leading to their unsatisfactory performance on class imbalanced data. For \(k\)NN, it takes the majority class in the \(k\) neighborhood of a test example as the classification result; this majority voting-based classification rule further degrades its performance on a class imbalanced problem. This is because the positive examples are usually sparse in the \(k\) neighborhood of a test example [6], i.e., most examples in the \(k\) neighborhood are usually negative examples; thus, the positive examples are often misclassified as negative ones by \(k\)NN, leading to the poor classification performance for positive examples. For instance, in the binary classification problem shown in Figure 1 (circles denote negative examples, triangles denote positive examples, and the cross denotes a test example), when \(k\) equals 7, there are 4 negative examples (N1-N4) and 3 positive examples (P1-P3) in the \(k\) neighborhood; obviously, \(k\)NN classifies the test example as the negative class although it actually belongs to the positive class.

Experiments conducted in Reference [13] indicate that SMOTE oversampling integrated with Random Undersampling (RUS) [14] or SMOTE oversampling integrated with the cost-sensitive MetaCost method [15] can both significantly improve the performance of C4.5 decision tree [2] on class imbalanced data. Unfortunately, these strategies do not work well for improving \(k\)NN in a class imbalanced scenario. The authors in [13] give the explanation from the following aspect: \(k\)NN makes a decision by investigating the local neighborhood of a test example, while the resampling and cost-sensitive strategies are global methods and are naturally inappropriate to \(k\)NN. Therefore, special methods for \(k\)NN need to be designed under the class imbalanced scenario.

As can be seen from the above illustration, improving \(k\)NN performance on imbalanced data is an important topic, which is of great significance to the expansion of its application fields and the enhancement of its practical utility. Over the past decade, researchers have conducted many works and proposed many methods. This paper tries to give a comprehensive survey of these works according to their perspectives and analyzes and compares their characteristics, which serves as a foundation for further study in this field.

The rest of this paper is organized as follows. The weighting strategy-based methods are illustrated in Section 2, the local geometrical structure-based methods are illustrated in Section 3, Section 4 introduces the fuzzy logic-based methods and followed by a category of methods based on missing positive example estimation in Section 5, Section 6 presents methods based on novel distance metrics while Section 7 presents dynamic-sized neighborhood-based methods, and conclusions and future work are presented in Section 8.

2. Methods Based on Weighting Strategy

This section introduces a category of methods that assigns weights to training examples in the neighborhood of a test example. In general, these methods can be divided into 5 subcategories as shown in the following.

2.1. Weighting Strategy Considering the Class Distribution around the Neighborhood. The authors in [12] claim that the reason for the unsatisfactory performance of \(k\)NN on the imbalanced data lies in the following: it only utilizes the local prior probabilities of each class in the neighborhood of a test example but does not employ the class distribution information around the neighborhood. In Figure 1, if the imbalanced class distribution around the test example’s neighborhood is considered, i.e., the area surrounded by the dotted rectangle, then the test example can be correctly classified as the positive class because in this dotted area, the positive nearest neighbors of the test example are much more than the negative ones. Therefore, the classification performance of \(k\)NN can be improved if such local class distribution information is utilized.

Based on the above observation, a weighting-based method is proposed in [12] to assign a test example-dependent local weight to each class, i.e., the examples’ weight in a class varies with the change of test examples rather than being a constant value. Concretely, for test example \(x_t \in \mathbb{R}^d\), the weight \(w_i^t\) of examples in class \(C_l\) (\(1 \leq l \leq L, L\) is the total number of classes in a classification problem) is calculated as follows. For the \(\lceil k/L \rceil\) number of nearest neighbors of \(x_t\) in class \(C_l\), if they are erroneously classified by traditional \(k\)NN, it is likely that these \(\lceil k/L \rceil\) neighbors belong to the minority class (the positive class) in the neighborhood of \(x_t\), in this case, the weight of class \(C_l\) is enlarged. Therefore,
the learned weights take into consideration the class distribution information around the neighborhood.

For the binary classification problem in Figure 1, \(L = 2\), when \(k\) equals 7, we have \(\lceil k/L \rceil = \lceil 7/2 \rceil = 4\); the test example’s 4 nearest neighbors in positive class \(C_1\) are P1 to P4, while its 4 nearest neighbors in negative class \(C_2\) are N1 to N4. It can be seen that P1, P2, and P4 are misclassified to be negative examples as the majority members of their 7 nearest neighbors are all negative examples; thus, the weight of these positive examples should be enlarged. Based on the enlarged weight of positive class, in the classification phase of 7NN for the test example, the 3 positive neighbors P1 to P3 have much larger weight than the 4 negative neighbors, i.e., P1 to P3 make more contribution to the classification result; thus, the correct classification is achieved.

However, the shortage of this weighting-based method is that approximately \(k\) (the exact number is \(\lfloor k/L \rfloor + 1\)) times of extra running of kNN is required around the neighborhood of each test example; thus, the computation cost is enlarged.

2.2. Weighting Strategy Based on Examples’ Informativeness. The authors in [16] believe that some examples carry more information than other examples: if an example is close to the test example and far from examples of other classes, then it is considered to be more informative. Following this idea, it is easily seen from Figure 2 that the example with index 2 carries more information than the one with index 1. The reason is that the two examples have roughly the same distance to the test example (the “query point” in Figure 2), but the example with index 1 is nearer to the class boundary, i.e., it is closer to the other classes. Based on the above consideration, the authors propose two informative kNN algorithms: the local information-based version LI-kNN and the global information-based version GI-kNN.

2.2.1. The Idea of LI-kNN. LI-kNN first finds the \(k\) nearest neighbors \(x_1^t, x_2^t, \ldots, x_k^t\) of test example \(x_t\) in the training set, then employs the designed metric to evaluate the informativeness of each training example in the \(k\) neighborhood, i.e., the evaluation scope is local, and selects the first \(I (I < k)\) most informative examples \(x_1^t, x_2^t, \ldots, x_I^t\). Finally, the majority class among the class labels \(y_1^t, y_2^t, \ldots, y_I^t\) of these \(I\) examples is regarded as the classification result. That is to say, the weight of the \(I\) selected examples is set to 1 while the weight of the other \((k-I)\) neighboring training examples is set to 0.

2.2.2. The Idea of GI-kNN. After LI-kNN determines \(I\) most informative neighbors \(x_1^t, x_2^t, \ldots, x_I^t\) and then makes a decision for test example \(x_t\), it separately determines the \(I\) most informative neighbors \(x_1^t, x_2^t, \ldots, x_I^t\) and does not utilize the informative neighbors \(x_1^t, x_2^t, \ldots, x_I^t\) of the previous test example \(x_t\). However, GI-kNN believes that some informative neighbors of a test example may be also the members of other test examples’ informative neighbors. For instance, two informative neighbors \(x_{1t}, x_{2t}\) of test example \(x_t\) may be members of the informative neighbor sets of three test examples \(x_{1t}, x_{2t}\) and \(x_{3t}\). In this case, training examples like \(x_{1t}, x_{2t}\) are considered to be global informative, thus assigning a larger weight to these examples. Based on the above idea, GI-kNN tries to find the training examples with global informativeness and assigns larger weights to them compared with ordinary examples.

To summarize, LI-kNN is a local strategy as it determines the informative examples in the \(k\) neighborhood of a test example. GI-kNN is a global strategy as it evaluates the informativeness of all the training examples and then assigns larger weights to the examples that are globally more informative, and besides, these weights are fixed when classifying all the subsequent test examples.

Experimental results indicate that GI-kNN and LI-kNN are not very sensitive to the change of parameter \(I\) and can achieve comparable performance with SVM. One drawback of GI-kNN is that the robustness of its adopted informativeness metric needs to be enhanced when there exist noisy examples in the training set.

2.3. Class Confidence-Based Weighting Strategy. The class confidence-weighted (CCW) kNN method [17] is proposed to assign weights to training examples in the neighborhood. As shown in Figure 3, the real boundary between the negative class (denoted by blue triangles) and the positive class (denoted by red circles) is represented by the solid blue line. There are 4 negative examples and 1 positive example in the \(k\) neighborhood of the test example (denoted by the solid green circle), and the nearest training example of the test example is a negative one. In this case, the classification result is certainly the negative class if the traditional majority voting-based classification rule is adopted. However, the test example actually belongs to the positive class and the negative examples in its neighborhood are also positive ones in reality. Thus, for each training example in the neighborhood, the probability that it belongs to its current class should be considered.

Based on this idea, for each training example \((x_j^t, y_j^t) = ((x_{j1}^t, x_{j2}^t, \ldots, x_{jk}^t), y_j^t) (j = 1, 2, \ldots, k)\) in the neighborhood of
test example \( x_t \), the confidence of its belonging to the current class \( y_t \) is calculated according to its attributes’ values \( (x_{t1}, x_{t2}, \ldots, x_{td}) \) and then this confidence serves as the weight \( w_t \) of this training example. For the instance shown in Figure 3, the mixture model and Bayesian network are employed in [17] to calculate the corresponding confidences of the 4 negative examples in the neighborhood and the values are 0.0245, 0.0173, 0.0171, and 0.0139, respectively. In addition, the calculated confidence of the positive example in the neighborhood is 0.1691. Obviously, the sum of the confidences of the 4 negative examples is much smaller than the confidence of the positive example, i.e., the weights of the 4 negative examples are much smaller than that of the positive example. In this way, the positive example in the neighborhood has much more influence on the classification result than the negative ones, ensuring that the test example can be correctly classified as a positive one. The conducted experiments also indicate that the class confidence-weighted method can correct \( k \)NN’s inherent bias to negative examples.

On the other hand, the class confidence-weighted method has to calculate the class confidence for each training example in the neighborhood, which increases the computation cost to some extent.

### 2.4. Weighting Strategy Based on Nearest Neighbor Density

A nearest neighbor density-based weighted class-wise \( k \)NN (WC\( k \)NN) algorithm is proposed in [18], and its basic idea is as follows.

First, the \( k \) nearest neighbor density of test example \( x_t \) is determined in each class.

This is implemented by constructing a \( k \) radius sphere \( S_{i,k}(x_t) \) that takes test example \( x_t \) as its center and contains at last \( k \) nearest examples from class \( C_l \) (\( 1 \leq l \leq L \)); then, the volume \( V_{i,k}(x_t) \) of this \( k \) radius sphere is used to denote the density: \( 1/V_{i,k}(x_t) \). It is not hard to see that, for a test example, its \( k \) radius sphere in the positive class usually has much larger volume than the one in the negative class due to the sparse distribution of positive class examples. As the radius of \( k \) radius sphere \( S_{i,k}(x_t) \) is determined by the distance \( d_{i,k}(x_t) \) between test example \( x_t \) and its \( k \)th nearest neighbor in class \( C_l \), thus \( d_{i,k}(x_t) \) is often used to approximately denote the volume of sphere \( S_{i,k}(x_t) \).

Second, the posterior probability of test examples belonging to each class is calculated based on the above \( k \) nearest neighbor density, which is shown in

\[
P(C_l | x_t) = \frac{\beta_l}{d_{i,k}(x_t)}, \quad l = 1, 2, \ldots, L. \tag{1}
\]

In formula (1), the weight \( \beta_l \) of class \( C_l \) is obtained by employing a certain convex optimization technique to optimize a nonlinear metric on the training set, and from this formula, we have the following observations. (a) For a class balanced data, two classes \( C_1 \) and \( C_2 \) have equal weights \( (\beta_1 = \beta_2) \); in this case, if examples in class \( C_1 \) are more densely distributed around test example \( x_t \), i.e., \( V_{1,k}(x_t) < V_{2,k}(x_t) \) and \( d_{1,k}(x_t) < d_{2,k}(x_t) \), then the probability of \( x_t \) belonging to \( C_1 \) is larger than that of \( C_2 \). (b) For an imbalanced data, compared with negative class \( C_n \), positive class \( C_p \) is more likely to be sparsely distributed around \( x_t \), i.e., \( d_{C_p,k}(x_t) > d_{C_n,k}(x_t) \); fortunately, the effect of this imbalanced distribution can be overcome by assigning larger weight for the positive class, i.e., \( \beta_p > \beta_n \).

At last, the class having the largest posterior probability is considered as the classification result: \( y = \arg \max_l P(C_l | x_t), \quad l = 1, 2, \ldots, L. \)

In terms of complexity, when classifying test example \( x_t \), WC\( k \)NN needs to run one time of \( k \)NN on each class to determine the \( k \) nearest neighbor density in this class. Thus, \( L \) (the total number of classes) times running of \( k \)NN are needed to classify a test example.

### 2.5. Weighting Strategy Integrated with Self-Adaptive \( k \)

The methods introduced above all use a constant \( k \) value, i.e., for each test example, \( k \) is the sum of the number of its positive neighbors and the number of its negative ones. Thus, the number of neighbors is not considered separately for each class.

To further improve the performance of weighted \( k \)NN methods, the authors in [19] propose to integrate the self-adaptive \( k \) technique with the example weighting strategy. In terms of weight determination, the positive examples are assigned larger weights than the negative ones; in terms of neighborhood size, the positive class is given small neighborhood size \( k_p \) while the negative class is given relative large neighborhood size \( k_n \), i.e., \( k_p > k_n \). In this way, the test example’s \( k_p \) positive neighbors and \( k_n \) negative neighbors constitute its neighborhood with size \( k_p + k_n \).

Accordingly, the classification result is determined by two aspects: (a) the weighted sum of the test example’s \( k_p \),
positive neighbors: $W_p(x_i) = \sum_{j=1}^{k} w_j$, and (b) the weighted sum of the test example’s $k_n$ negative neighbors: $W_n(x_i) = \sum_{j=1}^{k_n} w_j$. The class with a larger value is the corresponding decision result. To sum up, the self-adaptive k-based weighted kNN is simple and flexible. As to the formula used in the assignment of each class’s neighborhood size, more efforts need to be made to ensure that it is theoretically sound.

3. Methods Based on Local Geometric Structure of Data

An algorithm named class conditional nearest neighbor distribution (CCNND) is presented in [20], which alleviates the class imbalanced problem by using the local geometric structure of data, and its basic idea is as follows.

3.1. Calculating the k Nearest Neighbor Distances in Each Class. For each training example $x_m$ in class $C_i$ ($1 \leq i \leq L$), the distances to its $k$ nearest neighbors (without considering itself) in class $C_i$ are calculated: $d_{i,k}(x_m) = (\text{dist}(x_m, x^1_m), \text{dist}(x_m, x^2_m), \ldots, \text{dist}(x_m, x^l_m))$, where $\text{dist}(x_m, x^t_m)$ is the distance between $x_m$ and its $q$th ($1 \leq q \leq k$) nearest neighbor $x^t_m$ in class $C_i$.

3.2. Making Decisions Based on the k Nearest Neighbor Distances of Test Example in Each Class. First, for test example $x_t$, the distances to its $k$ nearest neighbors in class $C_i$ ($1 \leq i \leq L$) are calculated: $d_{i,k}(x_t) = (\text{dist}(x_t, x^1_t), \text{dist}(x_t, x^2_t), \ldots, \text{dist}(x_t, x^l_t))$, where $\text{dist}(x_t, x^q_t)$ is the distance between $x_t$ and its $q$th nearest neighbor $x^q_t$ ($1 \leq q \leq k$) in class $C_i$.

Second, for each class, the number of its training examples with larger $k$ nearest neighbor distances than the test example is determined: $N_i(x_t) = \{x_t \mid (x_t \in C_i) \land (d_{i,k}(x_t) > d_{i,k}(x_i)), l = 1, 2, \ldots, L, \text{where } d_{i,k}(x_m) > d_{i,k}(x_i) \text{ is equivalent to } (\text{dist}(x_m, x^1_m) > \text{dist}(x_t, x^1_t)) \land (\text{dist}(x_m, x^2_m) > \text{dist}(x_t, x^2_t)) \land \ldots \land (\text{dist}(x_m, x^l_m) > \text{dist}(x_t, x^l_t))\}$. It can be seen that the more such examples of a class have, the closer its class center is to the test example, i.e., the more likely the test example belongs to this class. Thus, the classification result is denoted as $y_t = \arg \max_{C_i \in \{1, 2, \ldots, L\}} N_i(x_t)$.

The conducted experiments demonstrate that, compared with the classical resampling and cost-sensitive methods, CCNND can achieve comparable or even better performance. As shown in Figure 4, the decision boundary obtained using CCNND is closer to the real boundary than that obtained using SVM and nearest neighbor. In addition, another advantage of CCNND is that it still works when the imbalance degree in a training set changes with time, e.g., in the case of online streaming data [21]. Therefore, CCNND can be applied in streaming data such as oil and natural gas industrial data.

4. Fuzzy Logic-Based Methods

In fuzzy logic-based [22] classification methods, the membership of belonging to each class is assigned to an example rather than a crisp class label, which can preserve abundant classification information and thus make a full classification. Based on this conclusion, a fuzzy weighted kNN algorithm is proposed in [22] by integrating the advantages of both fuzzy logic and weighted kNN, which is the first method introduced in Subsection 4.1, and the second method in Subsection 4.2 is a further improvement of fuzzy kNN itself.

4.1. Fuzzy Weighted kNN Algorithm. The fuzzy weighted kNN in [22] improves the weighted kNN method by utilizing the advantage of fuzzy logic, and it has the following three steps.

4.1.1. Determining the Class Membership of Each Example. The membership of example $x \in \mathbb{R}^d$ for class $C_l$ ($1 \leq l \leq L$) is calculated using formula (2), where $n_{C_l}$ is the number of training examples belonging to class $C_l$ in the $k$ neighborhood of example $x$ and $C(x)$ is the true class label of $x$.

\[
\mu_{C_l}(x) = \begin{cases} 
0.51 + \left( \frac{n_{C_l}}{k} \right) \ast 0.49, & \text{if } C_l = C(x), \\
\left( \frac{n_{C_l}}{k} \right) \ast 0.49, & \text{otherwise}. 
\end{cases}
\]  

For instance, in binary classification, if the true class of example $x_t$ is class $C_1$ and there are 4 neighbors belonging to class $C_1$ in its 5 neighborhood (i.e., $k = 5$), then the membership of $x_t$ for $C_1$ is $\mu_{C_1}(x_t) = 0.51 + (4/5) \ast 0.49 = 0.902$ while the one for $C_2$ is $\mu_{C_2}(x_t) = (1/5) \ast 0.49 = 0.098$.

4.1.2. Determining the Weight of Each Class. The weight $w_l$ of class $C_l$ ($1 \leq l \leq L$) is calculated using formula (3), where $N(C_l)$ denotes the number of examples in $C_l$, i.e., the size of
this class. It is easy to see that the positive class is assigned a weight of 1 while the negative class is assigned a weight less than 1, and the more examples in the negative class the smaller its weight.

\[ w_l = \frac{1}{(N(C_l)/\min \{N(C_1), N(C_2), \ldots, N(C_l)\})}, \quad l = 1, 2, \ldots, L. \]  

\[ \mu_{C_l}(x_l) = \frac{\sum_{l=1}^{k_l} w_j \cdot \mu_{C_l,j}(x_l)}{\sum_{j=1}^{k_l} w_j}, \quad l = 1, 2, \ldots, L. \]  

4.1.3. Making a Decision Based on the Class Memberships and the Class Weights. The class membership \( \mu_{C_l}(x_l) \) of test example \( x_l \) for class \( C_l \) (\( 1 \leq l \leq L \)) is calculated using formula (4), where \( \mu_{C_l,j}(x_l) \) is the class membership of \( x_l \)'s jth nearest neighbor for class \( C_l \) and \( w_j \) is the weight of the class to which this neighbor belongs.

At last, the decision is the class having the largest membership: \( y(x_l) = \arg \max_{l \in \{1, 2, \ldots, L\}} \mu_{C_l}(x_l) \).

4.2. Self-Adaptive k-Based Fuzzy kNN. Although the weighted fuzzy kNN introduced in Subsection 4.1 can achieve good performance, the fuzzy kNN algorithm itself cannot accurately compute examples’ class membership under the class imbalanced scenario. To solve this problem, an improved fuzzy kNN algorithm based on self-adaptive k strategy is proposed in [23] and it contains the following steps.

4.2.1. Determining the Neighborhood Size k for Each Class. The basic idea is to use relatively large neighborhood for the negative class and small neighborhood for the positive class. Concretely, the neighborhood size of each class is determined using formula (5). Where \( N(C_l) \) is the number of training examples in class \( C_l \), \( \lambda \) is a constant (e.g., take the value 1) with the purpose of preventing the value of \( k_{C_l} \) from being too small.

\[ k_{C_l} = \min \left\{ \lambda \cdot \frac{k \cdot N(C_l)}{\max \{N(C_l) \mid l = 1, 2, \ldots, L\}}, k, N(C_l) \right\}, \]  

where \( l = 1, 2, \ldots, L. \)

4.2.2. Calculating the Class Membership of Training Examples According to the Obtained Neighborhood Size. Formula (6) adopted here is different from formula (2): the corresponding class’s \( k \) value \( k_{C_l} \) is utilized when calculating the class membership of example \( x \), and \( C(x) \) is the true class label of \( x \).

\[ \mu_{C_l}(x) = \begin{cases} 0.51 + \left( \frac{n_{C_l}}{k_{C_l}} \right) \cdot 0.49, & \text{if } C_l = C(x), \\ \left( \frac{n_{C_l}}{k_{C_l}} \right) \cdot 0.49, & \text{otherwise}, \end{cases} \]  

where \( l = 1, 2, \ldots, L. \)

4.2.3. Determining the Class Membership of the Test Example. The class membership of test example \( x_l \) for class \( C_l \) (\( 1 \leq l \leq L \)) is calculated using formula (7), where \( \mu_{C_l}(x^l_j) \) is the class membership that example \( x^l_j \) belongs to class \( C_l \), and \( x_l^j \) is the \( j \)th \((1 \leq j \leq k_{C_l})\) neighbor of test example \( x_l \) in class \( C_l \). It can be seen from formula (7) that, in fact, test example \( x_l \)'s class membership for class \( C_l \) is the distance weighted sum of the corresponding class memberships of \( x_l \)'s \( k_{C_l} \) nearest neighborhoods.

\[ \mu_{C_l}(x_l) = \frac{\sum_{j=1}^{k_{C_l}} \mu_{C_l}(x^l_j) \left( \frac{1}{\left\| x_l - x^l_j \right\|^{2/(p-1)}} \right)}{\sum_{j=1}^{k_{C_l}} \left( \frac{1}{\left\| x_l - x^l_j \right\|^{2/(p-1)}} \right)}, \]  

where \( l \) takes values from \( \{1, 2, \ldots, L\} \) and \( p \) is an integer and is larger than 1.

At last, the class having the largest membership is the classification decision: \( y(x_l) = \arg \max_{l \in \{1, 2, \ldots, L\}} \mu_{C_l}(x_l) \).

By adopting different neighborhood sizes for different classes, this self-adaptive k-based fuzzy kNN can effectively alleviate the adverse influence of negative examples in the neighborhood of a positive example, making the obtained class membership more objective and thus improving the classification performance of fuzzy kNN on imbalanced data.

5. Methods Based on Missing Positive Data Estimation

The class imbalanced problem is regarded as a missing positive data estimation problem in [24]. From this perspective, a method called Fuzzy-based Information Decomposition (FID) is proposed and its main idea is as follows.

(1) \( t \) (\( t > 0 \)) number of synthetic positive examples are generated, and at the beginning, all their attributes have missing values

(2) The values of each attribute are estimated according to the current training set.

Concretely, for the \( s \)th (\( s = 1, 2, \ldots, d, d \) is the dimension of training data) attribute \( \text{attr}^s \):

(1) Dividing all the available values on \( \text{attr}^s \) into \( t \) intervals

According to the values of the current training set on \( \text{attr}^s \) (i.e., \( \text{attr}^s_1, \text{attr}^s_2, \ldots, \text{attr}^s_J \)) and the number \( t \) of positive examples to be generated, \( t \) intervals are obtained: \( q_1 = [a, a + h], \)
\[ q_m = (a + h, a + 2h], \ldots, q_t = (a + (t - 1) \cdot h, b], \] where \( \text{attr}_i \) is the value of the \( i \)-th \( (i = 1, 2, \ldots, N) \) training example on attribute \( \text{attr} \). \( N \) is the total number of current training examples, \( a \) and \( b \) are, respectively, the minimum and maximum values among \( (\text{attr}_1, \text{attr}_2, \ldots, \text{attr}_N) \), and \( h \) is the step length: \( h = (b - a)/t \).

(2) Generating a synthetic attribute value for each interval

For the \( m \)-th \( (1 \leq m \leq t) \) interval \( q_m \), a synthetic value \( \text{attr}_m \) of attribute \( \text{attr} \) is generated in the following way. The fuzzy membership \( \mu(x_i, q_m) \) of training example \( x_i \)'s \((i = 1, 2, \ldots, N)\) attribute value \( \text{attr}_i \) with respect to interval \( q_m \) is calculated, which is used as the weight \( w_m^i \) of example \( x_i \) in estimating the \( m \)-th missing value of attribute \( \text{attr} \). For instance, if the attribute value \( \text{attr}_i \) of example \( x_i (j = 1, 2, \ldots, N) \) is a “neighbor” to the center \( ((a + (m - 1) \cdot h) + (a + m \cdot h))/2 \) of interval \( q_m \), i.e., their distance is less than the step length \( h \), then the corresponding fuzzy membership \( \mu(x_i, q_m) \) is calculated and served as the weight \( w_m^i = \mu(x_i, q_m) \) of example \( x_i \); otherwise, the weight of example \( x_i \) is set to 0: \( w_m^i = 0 \). Therefore, the \( m \)-th estimated value for attribute \( \text{attr} \) can be represented in formula (8).

\[
\text{attr}_m = \frac{\sum_{j=1}^{N} w_m^j \cdot \text{attr}_j}{\sum_{j=1}^{N} \mu(x_j, q_m) \cdot \text{attr}_j}. \tag{8}
\]

The weights satisfy \( \sum_{j=1}^{N} w_m^j = 1 \).

That is to say, only when an example’s attribute value is close to the center of interval \( q_m \) \((m = 1, 2, \ldots, t)\) can it effectively influence the calculation of the \( m \)-th synthetic attribute value \( \text{attr}_m \). Thus, the \( m \)-th estimated value for attribute \( \text{attr} \) is the weighted sum of these effective training examples on this attribute.

The advantage of FID is that it can deal with data with arbitrary dimension as it separately generates the missing values for each attribute. Traditional methods like Random OverSampling (ROS) [14] and Clustering Based OverSampling (CBO) [25] have the tendency of overfitting due to the replication of existing positive examples; for methods like SMOTE [26] and Majority Weighted Minority (MWM) oversampling [27], an approximate positive example needs to be selected before generating a synthetic positive example using linear interpolation. However, these traditional methods have poor performance when the positive examples in the original training set are not enough. Fortunately, FID can overcome this problem as it generates the synthetic values separately for each attribute. As to the disadvantage of FID, when calculating a synthetic value for an attribute, the memberships of all training examples’ values on this attribute with respect to the current interval need to be computed, leading to high computation and time complexity in the case of large training set size.

6. Novel Distance Metric-Based Methods

Euclidean distance is usually adopted as the metric to evaluate the similarity between two examples. However, this metric does not treat the positive and negative examples separately in the calculation of distance. To make up this shortcoming, the following works, respectively, present a novel distance metric that is sensitive to positive examples.

6.1. Distance Metric for Exemplar Positive Examples. The authors in [28] propose a method called \( k \) exemplar-based Nearest Neighbor (k-ENN), which improves the classification performance to positive examples by extending each exemplar positive example from a point in the feature space to a Gaussian ball. In detail, the principle of k-ENN is as follows.

6.1.1. Determining the Exemplar Positive Examples. For each positive example \( x_i^\tau (1 \leq i \leq n_p) \), \( n_p \) is the total number of positive examples in the training set) in training set \( D_{tr} \), find its nearest positive class neighbor \( x_j^\tau (i \neq j) \) and compute their distance \( r_i^\tau = \text{dist}(x_i^\tau, x_j^\tau) \). In this way, a Gaussian ball centered at \( x_i^\tau \) and with a radius of \( r_i^\tau \) is constructed, i.e., a neighborhood \( S_i \) of \( x_i^\tau \) is obtained. As \( x_i^\tau \) is the nearest neighbor of \( x_j^\tau \) in positive class, there are only two positive examples in Gaussian ball \( S_i \): \( x_i^\tau \) and \( x_j^\tau \), and the other negative examples occurring in \( S_i \) are “false positives.” k-ENN considers positive example \( x_i^\tau \) as an exemplar positive example if the false positive rate \( f_p_i \) in its Gaussian ball \( S_i \) is less than a threshold: \( f_p_i < \tau \).

For instance, Figure 5 displays the Gaussian balls of three positive examples numbered 1 to 3, which are denoted using dashed circles. It is easy to see that there is no negative example (denoted by symbol “.”) in each Gaussian ball; thus, the three positive examples are all exemplar ones.

6.1.2. Defining the Distance Between the Test and Training Examples. When classifying test example \( x_t \), its distance to each training example \( x_i (1 \leq i \leq N) \) needs to be computed.
6.2. Distance Based on Examples' Weights. The example weighting-based distance is proposed in [29]; it considers the relative importance of each training example \( x_i \) and the positive example or a negative one, then the distance is still the Euclidean distance:

\[
d(x_i, x_j) = \text{dist}(x_i, x_j). \tag{10}
\]

The distance in formula (9) subtracts the radius of the Gaussian ball of exemplar positive example \( x_i \); in reality, it is the distance between test example \( x_t \) and the boundary of \( x_i \)'s Gaussian ball. In this way, the distance from exemplar positive examples to the test example is reduced such that these exemplar positives are given more attention in the classification phase, consequently improving the classification performance for positive examples.

6.2. Distance Based on Examples' Weights. The example weighting-based distance is proposed in [29]; it considers the relative importance of each training example \( x_i \) (1 ≤ \( i \) ≤ \( N \)) when classifying test example \( x_t \), which is implemented by utilizing training examples' weights rather than simply computing the Euclidean distance:

\[
d_w(x_t, x_i) = \frac{\|x_t - x_i\|}{w_i}. \tag{11}
\]

It can be seen from formula (11) that, by assigning larger weights for positive training examples than for negative ones, the distances from positive examples to the test example can be reduced, which has the effect of improving the chance of positive examples being selected into the neighborhood, thus rectifying the inherent bias of \( kNN \) for negative examples and improving the classification performance of positive examples. As to the weights \( w_i (i = 1, 2, \ldots, N) \) assigned to the training examples, they are obtained by employing a gradient ascend technique to optimize the G-mean [30] metric.

7. Methods Based on Dynamic-Sized Neighborhoods

The methods introduced above construct the neighborhoods with equal size for all the test examples: traditional \( kNN \) constructs the neighborhood by finding \( k \) nearest training examples for a test example, while self-adaptive \( k \) strategy-based \( kNN \) constructs the neighborhood by finding \( k_p \) positive neighboring examples and \( k_n \) negative neighbors for each test example, and in a classification problem, the number \((k_p + k_n)\) of nearest neighbors does not change with respect to test examples. Different from these methods, the methods illustrated in this section construct test example-dependent neighborhood sizes, i.e., for a classification problem, the neighborhood size changes with different test examples. The aim of doing so is to ensure the existence of sufficient positive examples in the neighborhood, and the positive examples in the neighborhood of a test example are all closer to the test example and have similar posterior class probabilities to it. Based on whether parameters are required in determining the dynamic neighborhood size, this kind of method can be divided into two categories.

7.1. Dynamic-Sized Neighborhood \( kNN \) with Parameters

7.1.1. Positive-Biased Nearest Neighbor \( PNN \). The Positive-biased Nearest Neighbor (PNN) algorithm is designed in [31] to improve the sensitivity of \( kNN \) to positive class. \( PNN \) first dynamically constructs the neighborhood of a test example and then adjusts the classification result according to the local class distribution in the neighborhood.

1. Constructing the “\( k/2 \) Positive Nearest Neighborhood (\( k/2-PNN \))” for each test example to expand the neighborhood for decision making.

Compute the distances from test example \( x_t \) to all the training examples and rearrange the training examples in ascending order according to their distances, then find the \( [k/2] \) number of positive nearest neighbors of \( x_t \) that constitute the neighborhood \( S^p_{k/2}(x_t) \). \( S^p_{k/2}(x_t) \) usually contains more examples than \( x_t \)'s \( k \) neighborhood \( S_k(x_t) \), i.e., \( r = |S^p_{k/2}(x_t)| > k \). As displayed in Figure 6(a), the test example is denoted using symbol “*” and the positive and negative examples are denoted using symbols “+” and “−”, respectively. When \( k = 5 \), the neighborhood having \( [k/2] = [5/2] = 3 \) positive training examples contains \( r = 8 \) examples in total, in this case \( r > k \), and an “extended neighborhood” \( S^p_{k/2}(x_t) \) is obtained for test example \( x_t \).

2. Making decision based on whether “\( k/2-PNN \)” is a positive subconcept
If the ratio of positive examples in \( S^p_{k/2} (x_t) \) is much higher than that in the overall training set \( D \), then \( S^p_{k/2} (x_t) \) is considered to be a positive class subconcept and \( [k/2]/k > 1/2 \) is set as the posterior probability of the test example for the positive class, i.e., \( x_t \) is classified as a positive class example. Otherwise, if \( S^p_{k/2} (x_t) \) is not a positive subconcept, then the ratio (i.e., \( [k/2]/r \)) of positive examples in \( S^p_{k/2} (x_t) \) is regarded as the posterior probability of \( x_t \) for the positive class; in this case, the probability is usually less than 0.5, i.e., \( x_t \) is classified as a negative example.

For instance, if the neighborhood in Figure 6(a) is a positive class subconcept, then the probability of the test example with respect to the positive class is \( P(+|x_t) = [k/2]/k = 3/5 > 1/2 \); thus, the classification result of PNN is the positive class; otherwise, the probability is \( P(+|x_t) = [k/2]/r = 3/8 < 1/2 \); thus, the test example is classified as a negative example. For the case that there are less than \( k \) examples in \( S^p_{k/2} (x_t) \) as shown in Figures 6(b) and 6(c), which rarely occurs under the class imbalanced scenario, it indicates that the positive examples are densely distributed around the test example and the corresponding probability is \( P(+|x_t) = [k/2]/r = 3/4 > 1/2 \), i.e., the decision of PNN is the positive class.

For the case that \( S^p_{k/2} (x_t) \) has the same size with \( k \) neighborhood \( S_p(x_t) \) as shown in Figure 6(d), PNN degrades to kNN.

Experiments in [31] indicate that the simple and effective decision bias of PNN can better classify positive examples, and PNN usually outperforms \( k \)-ENN [28] and achieves comparable performance with CCW-kNN [17] mentioned in previous sections. In terms of efficiency, PNN has a much lower computation cost than the two methods that require a "training phase": (1) \( k \)-ENN needs to determine all the exemplar positive examples in the training phase to expand its decision boundary while (2) CCW-kNN computes the weight of each training example by using the mixture model and the Bayesian network. Therefore, both the two methods have a high computation cost. In addition, PNN also outperforms oversampling techniques like SMOTE as well as costsensitive strategies like MetaCost.

### 7.2. Dynamic-Sized Neighborhood kNN without Parameters

#### 7.2.1. Gravitational Fixed Radius Nearest Neighbor (GFRNN)

The methods introduced above often have many parameters and complex structure [33], leading to their high time complexity. These methods include the class weighted kNN [12], the examples’ informativeness-based kNN (e.g., LI-kNN [16]), the class confidence-weighted kNN [17], the class conditional nearest neighbor distribution (CCNND) [20] as well as \( k \)-ENN [28], and PNN [31] in the previous subsection. In addition, the global information is not fully utilized in these methods.

To overcome these drawbacks, a gravitational fixed radius nearest neighbor (GFRNN) algorithm is proposed in [34], which is inspired by the concept of gravitation in classical dynamics. GFRNN is formed by introducing the "gravitation between example pair" into the fixed radius nearest neighbor method. Concretely, GFRNN operates as follows:

1. The distance between each example pair is first calculated, and then, their average value is adopted as the neighborhood radius \( R \) of a test example, which is shown in

   \[
   R = \frac{1}{2} N (N - 1) \sum_{x_i, x_j \in D} \text{dist}(x_i, x_j). \tag{12}
   \]

   In formula (12), \( D \) denotes the training set. Thus, the neighborhood of test example \( x_t \) can be described in formula (13), which is constituted by training examples having a distance no more than \( R \) with \( x_t \).

   \[
   S(x_t) = \{ x_i \mid (x_i \in D) \land (\text{dist}(x_i, x_t) \leq R) \}. \tag{13}
   \]

2. Computing the gravitation between each training example in the neighborhood and the test example, which can be achieved using

   \[
   f(x_i, x_t) = G \frac{m_{x_t} m_{x_i}}{d(x_i, x_t)^3}, \quad i = 1, 2, \ldots, |S(x_t)|. \tag{14}
   \]

   To simplify the computation, both the gravitational constant \( G \) and the mass \( m_{x_t} \) of test example \( x_t \) are set to 1, where mass \( m_{x_t} \) is essentially the weight of test example \( x_t \). Thus, only the mass \( m_{x_i} \) of training example \( x_i \) in the neighborhood needs to be determined. To balance the effects of the positive
and negative examples in the neighborhood, the mass of each training example is calculated using formula (15), where $D_p (D_n)$ denotes the set of positive (negative) examples in training set $D$. It is easy to see that the mass (weight) of positive examples in the neighborhood is the ratio between the number of negative examples and that of positive examples in training set $D$, i.e., the class imbalance ratio IR. In addition, the mass (weight) of negative examples is set to 1 in GFRNN.

$$m_j = \begin{cases} \text{IR}, & \text{if } (x_i \in S(x_j)) \land (x_j \in D_p), \\ 1, & \text{if } (x_i \in S(x_j)) \land (x_j \in D_n). \end{cases}$$  \hspace{1cm} (15)

(3) Making a decision according to the gravitation of training examples in the neighborhood to the test example

$$F(x_t) = \frac{n_{pos}(x_i)}{\sum_{j=1}^{n_{pos}} d(x_i, x_j)} - \frac{n_{neg}(x_i)}{\sum_{j=1}^{n_{neg}} d(x_i, x_j)},$$  \hspace{1cm} (16)

where $n_{pos}(x_i)$ and $n_{neg}(x_i)$ in formula (16) are, respectively, the numbers of positive and negative examples in the neighborhood $S(x_i)$ of test example $x_t$. Formula (16) indicates that GFRNN makes a decision in this way: if the gravitation sum of positive examples in $S(x_i)$ is larger than that of the negative examples, then $x_t$ is classified to be a positive example; otherwise, $x_t$ is classified as a negative example.

It can be learned from the above illustration that, in determining the neighborhood for a test example, GFRNN does not require any parameter and only utilizes this global information: the average distance among training example pairs. In addition, another global information, i.e., the class imbalance ratio IR in the training set, is used as the weight of positive neighbors. To sum up, GFRNN has the following advantage: it can effectively address the class imbalanced problem and does not require the initialization or adjustment of any parameters, which further expands the family of kNN classification algorithms. On the other hand, GFRNN only employs the overall class imbalance ratio IR in the training set to set weights for positive neighbors but does not utilize any local information concerning training examples, which can be seen as its disadvantage. To solve this problem, the following two works, respectively, present a solution.

7.2.2. Two Improvement Algorithms for GFRNN

(1) The First Improvement Algorithm. An entropy and gravitation-based dynamic radius nearest neighbor (EGDRNN) is proposed in [35]; its differences with GFRNN mainly lie in the following two aspects.

(a) Neighborhood radius determination

EGDRNN determines the radius of test example $x_t$’s neighborhood by first computing its average distance $\text{avgdist}_{D_p} (x_t)$ to the positive examples $D_{pos}$ in training set $D$ and its average distance $\text{avgdist}_{D_n} (x_t)$ to the negative examples $D_{neg}$ in training set $D$, respectively, and then taking the sum of these two values as the neighborhood radius, which is shown in formula (17).

$$R_{x_t} = \text{avgdist}_{D_p} (x_t) + \text{avgdist}_{D_n} (x_t) = \frac{1}{n_{pos}} \sum_{i=1}^{n_{pos}} \text{dist}(x_i, x_t) + \frac{1}{n_{neg}} \sum_{j=1}^{n_{neg}} \text{dist}(x_i, x_j).$$  \hspace{1cm} (17)

Therefore, the radius determined by EGDRNN for a test example depends on the location of test example with respect to the positive and negative classes and varies with different test examples.

(b) Weighting strategy for examples in neighborhood

In addition to IR, EGDRNN also introduces the information entropy concept to make examples in different locations have different degrees of importance. Concretely, for training example $x_i, \quad (i = 1, 2, \cdots, |S(x_i)|)$ in neighborhood, EGDRNN computes its information entropy $E(x_i)$ using formula (18). $C_1$ and $C_2$ denote the positive and negative classes, respectively; $p(x_i, C_1)$ denotes the probability of example $x_i$ belonging to the positive class while $p(x_i, C_2)$ denotes the probability of example $x_i$ for the negative class, where probability $p(x_i, C_1)$ is calculated using the proportion of positive examples in the $k$ neighborhood of example $x_i$ while $p(x_i, C_2)$ is calculated using the corresponding proportion of negative examples. It can be seen that the smaller the information entropy of $x_i$ the higher the certainty degree of its belonging to a certain class; otherwise, the larger the information entropy, the lower the certainty degree, i.e., it is closer to the decision boundary.

$$E(x_i) = -p(x_i, C_1) \ln p(x_i, C_1) - p(x_i, C_2) \ln p(x_i, C_2).$$  \hspace{1cm} (18)

To sum up, for a test example, the gravitation sum of examples in its neighborhood is calculated as follows:

$$F(x_t) = \frac{n_{pos}(x_t)}{\sum_{i=1}^{n_{pos}} \text{IR} \cdot E(x_i)} - \frac{n_{neg}(x_t)}{\sum_{j=1}^{n_{neg}} E(x_j)}. $$  \hspace{1cm} (19)

Formula (19) demonstrates that EGDRNN pays more attention to the positive examples in the neighborhood as well as the examples that are close to the class boundary. Experimental results indicate that EGDRNN not only achieves a high classification accuracy but also has the lowest time cost among the comparison algorithms.

(2) The Second Improvement Algorithm. The improvements made in [33] lie in the following aspects. When determining
the weight for neighboring training example $x_i$ ($i = 1, 2, \ldots, |S(x_j)|$), in addition to IR, the gravitation from other examples $x_i$ ($x_i \in D, x_i \neq x_j$) to the current example $x_j$ is also considered to calculate its weight. The authors in [30] believe that, for a training example, the larger the sum of gravitation from other examples, the denser its surrounding examples (local information), and they regard such training example as unimportant and assign relatively low mass (weight) for it. In this way, both the global example information (i.e., IR) and this local information are utilized in determining the weights for neighbors.

8. Conclusion

$k$NN is a simple and effective base learning algorithm and can achieve comparable classification performance with more complex classifiers such as decision tree and artificial neural networks. However, $k$NN does not work well on imbalanced data due to the usage of overall classification accuracy as its optimization objective as well as its majority voting-based classification rule. To solve this problem, researchers have conducted many works and proposed a lot of solutions. This paper gives a comprehensive survey of these works according to their adopted perspectives and analyzes and compares their characteristics. What is more, there are still some problems that deserve further study in this field. For instance, we list three of them in the following:

1. Most algorithms introduced in this paper mainly consider the case that there is only one positive class in imbalanced data; thus, in the case of two or more positive classes, how to adjust these algorithms to make them work is an important problem

2. For the global information-based algorithm GI-$k$NN, how to improve the robustness of its adopted information metric to noisy training examples needs to be investigated

3. For the online streaming data in which the class imbalance degree can change with time, only the class conditional nearest neighbor distribution algorithm CCNND introduced in Section 2 is applicable to this scenario. Thus, more efforts need to be made to make other methods introduced in this paper also suitable to online streaming data

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is supported by the Natural Science Foundation of Shandong Province (ZR2018QF002).

References


